

**AFRL-VA-WP-TR-2006-3074**

**AIR VEHICLE TECHNOLOGY  
INTEGRATION PROGRAM (AVTIP)**

**Delivery Order 0008: Open Control Platform  
(OCP) Software Enabled Control (SEC) Hardware  
in the Loop Simulation Program**

**Eric Portilla**

**Northrop Grumman Corporation  
One Hornet Way  
El Segundo, CA 90245**



**JULY 2004**

**Final Report for 01 October 2001 – 28 May 2004**

**Approved for public release; distribution is unlimited.**

**STINFO COPY**

**AIR VEHICLES DIRECTORATE  
AIR FORCE MATERIEL COMMAND  
AIR FORCE RESEARCH LABORATORY  
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542**

# NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Wright Site (AFRL/WS) Public Affairs Office (PAO) and is releasable to the National Technical Information Service (NTIS). It will be available to the general public, including foreign nationals.

PAO Case Number: AFRL/WS 06-1261, 10 May 2006.

THIS TECHNICAL REPORT IS APPROVED FOR PUBLICATION.

//Signature//

---

STANLEY H. PRUETT  
Aerospace Vehicles Technology  
Assessment & Simulation Branch

//Signature//

---

GARY K. HELLMANN, Chief  
Aerospace Vehicles Technology  
Assessment & Simulation Branch

//Signature//

---

JEFFREY C. TROMP  
Senior Technical Advisor  
Control Sciences Division  
Air Vehicles Directorate

This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. <b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b></p>						
1. REPORT DATE (DD-MM-YY) July 2004		2. REPORT TYPE Final		3. DATES COVERED (From - To) 10/01/2001– 05/28/2004		
4. TITLE AND SUBTITLE AIR VEHICLE TECHNOLOGY INTEGRATION PROGRAM (AVTIP) Delivery Order 0008: Open Control Platform (OCP) Software Enabled Control (SEC) Hardware in the Loop Simulation Program				5a. CONTRACT NUMBER F33615-00-D-3054-0008		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER 0609199		
6. AUTHOR(S) Eric Portilla				5d. PROJECT NUMBER A0D1		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER 0A		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)  Northrop Grumman Corporation One Hornet Way El Segundo, CA 90245				8. PERFORMING ORGANIZATION REPORT NUMBER  NOR 04-801		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)  Air Vehicles Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson Air Force Base, OH 45433-7542				10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL-VA-WP		
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-VA-WP-TR-2006-3074		
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES Report contains color. PAO Case Number: AFRL/WS 06-1261, 10 May 2006.						
14. ABSTRACT  These annotated Final Review charts summarize the work performed for the Open Control Platform (OCP) Software Enabled Control (SEC) Hardware in the Loop (HITL) Delivery Order #0008 under the Air Vehicles Technology Integration Program (AVTIP) contract awarded to Northrop Grumman Corporation (NGC). The OCP HITL program developed a Hardware-in-the Loop facility for demonstrating and evaluating High-Confidence Software and Systems (HCSS). Boeing, AFRL, and NGC created and implemented an architecture that provided AFRL/VAC with the baseline capability to test control algorithms (including collision avoidance and Fault Detection Isolation), run combined piloted and autonomous vehicle simulations, and created an interface to AFRL visualization software Virtual Battlefield Management System (VBMS) and SubrScene.						
15. SUBJECT TERMS Flight Simulation, Hardware-in-the-Loop Simulation, Infinity Cube Simulator, Middleware, Open Control Platform, Software Enabled Control						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT: SAR	18. NUMBER OF PAGES 44	19a. NAME OF RESPONSIBLE PERSON (Monitor) Stanley H. Pruett 19b. TELEPHONE NUMBER (Include Area Code) N/A	
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified				



#### Abstract:

These annotated Final Review charts summarize the work performed for the Open Control Platform (OCP) Software Enable Control (SEC) Hardware in the Loop (HITL) Delivery Order #0008 under the Air Vehicles Technology Integration Program (AVTIP) contract awarded to Northrop Grumman Corporation (NGC). The OCP HITL program developed a Hardware-in-the-Loop facility for demonstrating and evaluating High-Confidence Software and Systems (HCSS). Boeing, AFRL, and NGC created and implemented an architecture that provided AFRL/VAC with the baseline capability to test control algorithms (including collision avoidance and Fault Detection Isolation), run combined piloted and autonomous vehicle simulations, and created an interface to AFRL visualization software Virtual Battlefield Management System (VBMS) and SubrScene.



OCP Hardware-in-the-Loop  
Demonstration Program

# Northrop Grumman Final Review



## Presentation Outline

### ■ Program Summary

#### ■ Program Results/Lessons Learned

- Scenario 1 - 1-On-1 Non-Cooperating
- Scenario 2 - 1-On-1 Cooperating
- Scenario 3 - Human-in-the-Loop

#### ■ Final Simulation Demonstration

- Scenario 4 - Formation Flight (Pack Formation)
- Scenario 5 - Formation Flight (S-Curve)
- Scenario 6 - Formation Flight With Collision Avoidance
- Scenario 7 - Formation Flight Random Fault Triggering

#### ■ Path Forward



## Program Objective

**Support AFRL/VAC in Developing a Hardware-in-the-Loop (HITL) Facility for Demonstration and Testing.**

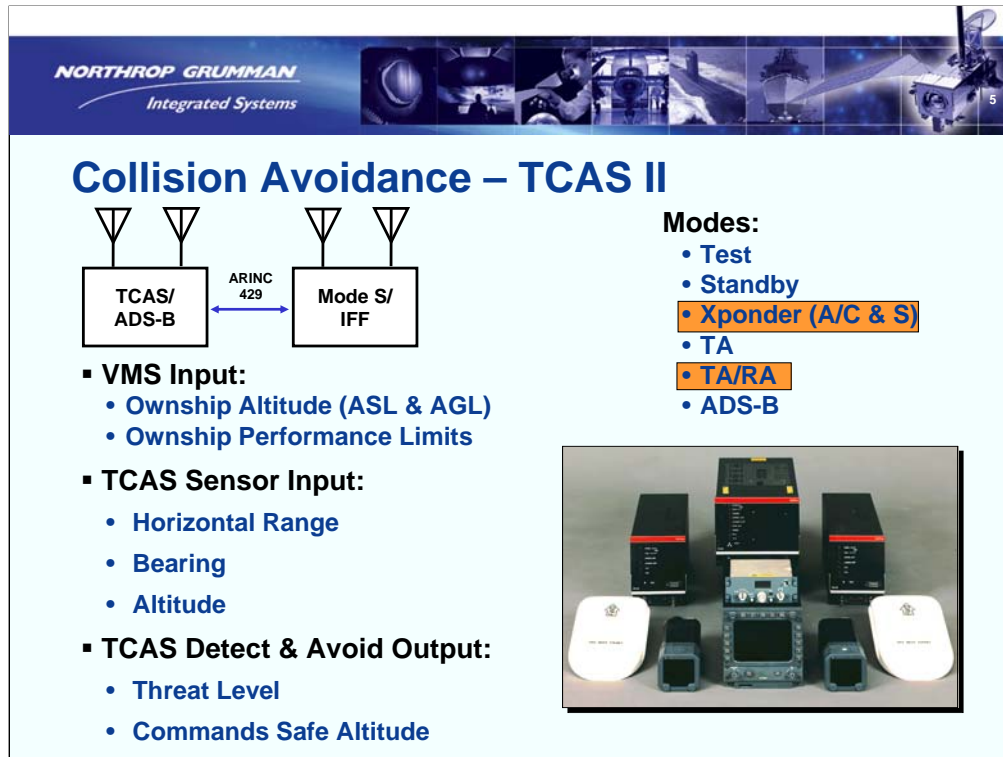
**Develop Control and Failure Detection Isolation Reconfiguration (Application Level) Algorithms Necessary to Support the Demos.**

- Collision Avoidance Algorithm and Its Associated Primitive Maneuvers
- 4D Waypoint Following
- Limited Mission/Pack Manager (Station Keeping Algorithm)
- Limited Fault Manager

**Develop Real Time Software with OCP on Windows Platform and Implement on a RT Linux/Hardware-in-the-Loop Environment.**

**Create OCP Architecture Inductive to Testing New Vehicle**

The objective of this program was to support AFRL/VAC in developing a Hardware-in-the Loop (HITL) facility for demonstrating and evaluating High-Confidence Software and Systems (HCSS). We as a team (Boeing, AFRL, and NGC) created and implemented an architecture that provides AFRL/VAC with the baseline capability to test control algorithms (including collision avoidance and FDIR), run combined piloted and autonomous vehicle simulations, and create visualization through VBMS and Subscene. To support these demonstrations NGC provided a UAV model as well as a Generic Tactical Fighter (GTF) model along with an inner loop controller for each. The outer loop controller algorithms were developed with generality in mind for use with both types of vehicles creating a uniform architecture for all vehicles. These algorithms were integrated into the OCP middleware environment on a Windows OS platform and handed off to Boeing who then ported the developed simulation into Linux and VxWorks on a PowerPC board for the hardware in the loop demonstration.

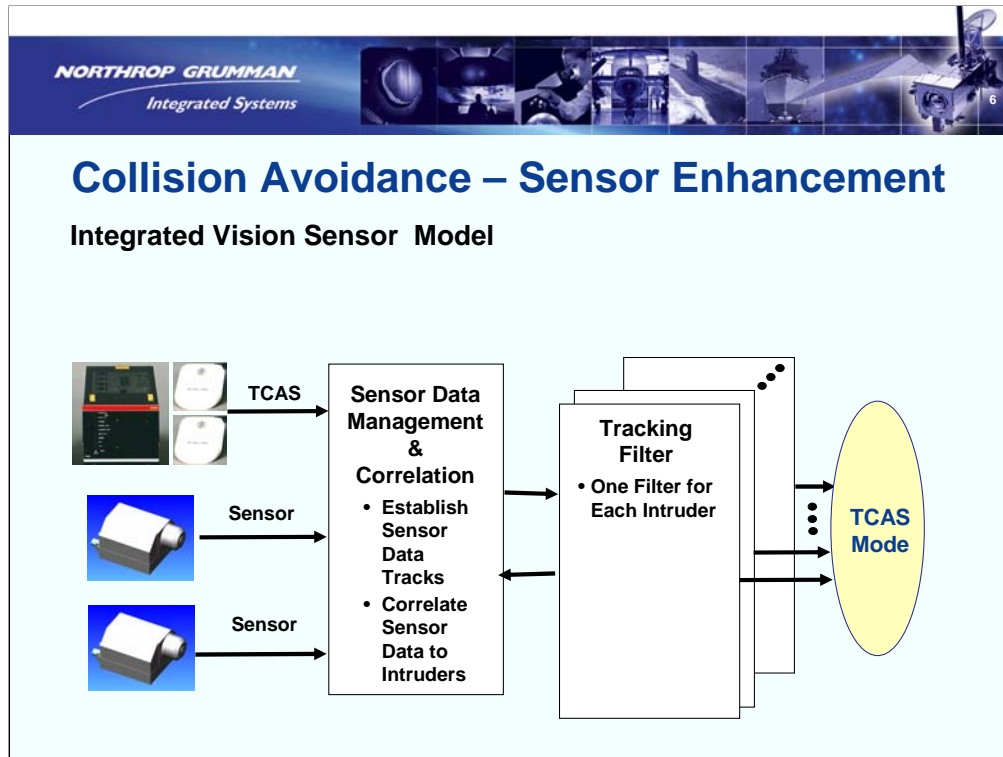


Traffic Alert/ Collision Avoidance System II was established as the baseline for all vehicles collision avoidance. The HITL TCAS model was initially developed from Pseudo-Code and evolved throughout the program to accommodate autonomous control, cooperative maneuvering, and specific vehicle envelope limitations (i.e. ceiling/floor).


TCAS consists of two main functions. The first function is to communicate positions between vehicles. Each vehicle broadcasts its altitude, determines range and bearing from the return signal. This data is then used by the second function consisting of a collision avoidance algorithm. This algorithm produces a suggested rate of climb to resolve the impending collision scenario.


The first version of TCAS modeled focused solely on the collision avoidance algorithm. The model assumed all vehicle states were known, removing the need to infer velocities and body attitudes, and emphasized converting a passive TCAS alerting system into an active autonomous maneuver command. Delays and filters were used to avoid alerts cycling on and off and to smooth commands. For the second version, the dive/climb logic was updated to accommodate cooperative collision avoidance. The final version was modified to more accurately represent the sensor side of TCAS. Only relative range, relative altitude, and bearing were available and tracked to estimate all unknown states.





The original vehicle architecture specified an OCP component for situational awareness sensors. This component originated as a second source of the communicated vehicle data with added sensor noise. By creating a second source of vehicle data the Fault Detection Isolation algorithm could detect signal failures and compensate for them. To demonstrate the versatility of OCP the final demonstration integrated simple vision algorithms. These algorithms converted absolute position received from the communication link to range, elevation, and azimuth angles and signal strength based on relative distance and vehicle size. With this added feature, vehicles could be initialized to transmit data to be passed via the communication link or if not available, then sensor data from these vision models would be available to analyze the data for input into the collision avoidance algorithm.






## Outer Loop Controls

- **Pack Controller Has Two Modes**
  - Waypoint Following
  - Station Keeping
  - Mode Switching Occurs
    - Loss of Communication With Leader
    - TCAS Failure in Leader
    - External Source (Mission Objectives Change)
- **Waypoint Guidance**
  - North, East, Altitude, Time (4D)
  - Commands Bank, Altitude, and Velocity
- **Station Keeping (SEC Developed)**
  - Follower Tracks Translating waypoint (Leader Position)
  - Follower Tracks With Longitudinal, Lateral And Altitude Offset
  - Commands Gamma, Chi, and Velocity

The Outer Loop controller developed consisted of two pack controller modes which commanded one guidance routine. The first mode was a 4-D waypoint follower developed for the initial 1 on 1 non-cooperating scenarios which tracked North, East, Altitude, and time by commanding bank angle, altitude, and velocity. The second mode was a station keeper developed for formation flight scenarios. The station keeper algorithm tracks the lead vehicles position with a longitudinal, lateral, and altitude offset. The station keeper commanded gamma, chi, and velocity. The difference in the two modes' commands created a need for modification to the architecture. An additional signal was added to the OCP Component Info file and the environment was regenerated using the Application Programming Interface (API) backend tool. The guidance routine will take in the data sent from the pack controller and, depending on the mode, run the correct method to return the commands to drive the inner loop controller (alpha, beta, mudot, and throttle).

The vehicle's outer loop control mode was determined during initialization whether the vehicle was designated a leader or a follower. Once the scenario began, mode switching would occur when the Fault Detection Isolation (FDI) detected any one of multiple faults. These faults included loss of communication with the leader, loss of TCAS in the leader, and/or erratic leader behavior. In the final scenario the only fault demonstrated was the detection of a hard over control surface failure in the lead vehicle.

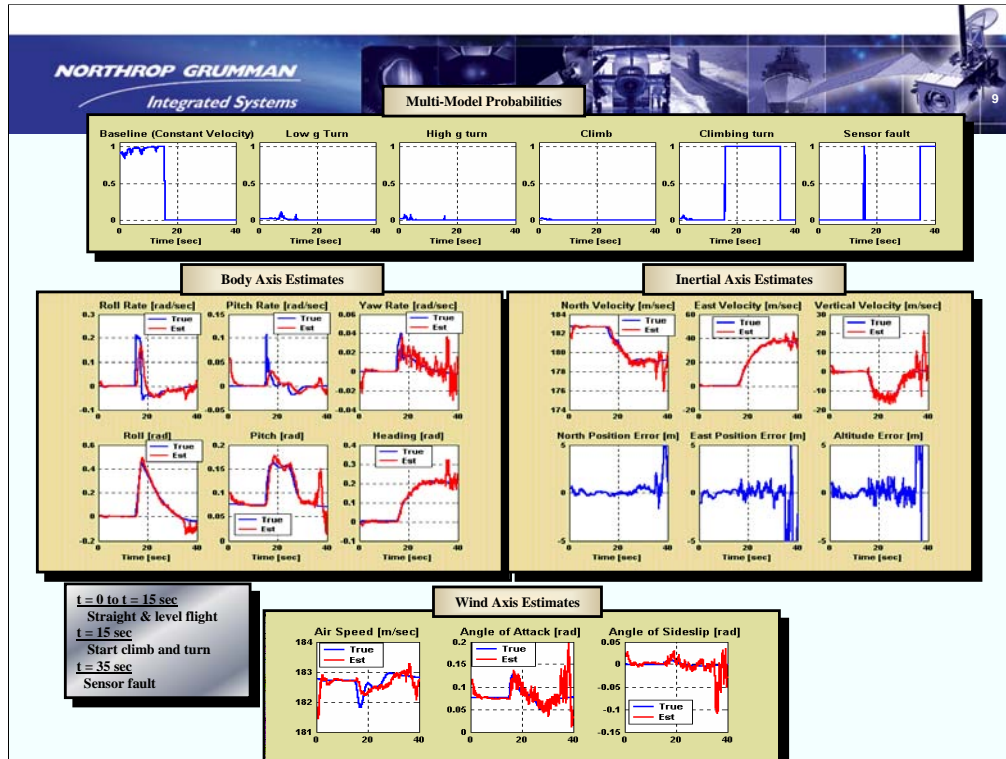
**NORTHROP GRUMMAN**  
*Integrated Systems*



## Kalman Filter FDI Description (SEC)

- **Filter tasks:**
  - Optimally fuse asynchronous vision and communications data
  - Infer velocities and body rates from noisy position and attitude measurements (Cat 3)
  - Provide smoothed position and attitude estimates
  - Reject bad data, detect sensor or leader failures
  - React to unexpected maneuvers
- **Filter uses attitude information to predict vehicle maneuvers**
  - Bank angle is measure of turning rate
    - Filter mechanism not restricted to level turns
  - Pitch angle combined with flight-path angle gives angle of attack
    - Indicates changes in lift and vertical acceleration
    - Aero properties of target vehicle assumed unknown
    - Filter estimates lift coefficient ( $CL\alpha$ ) and trim angle of attack
    - Deviations from simplified lift equation modeled as random acceleration (Markov process)

The Kalman filter used for FDI was developed under the DARPA SEC program. The filter inputs data from two sources. The first data set sent via the communication link includes position, attitude, and velocity information while the second source of data originated from the situational awareness sensors and consisted of position data only. The filter receives these two independent data sets and provides smoothed position, velocity, and attitude estimates while removing erroneous data and flagging sensor or communication failures. This smoothed filtered data is then compared to predefined vehicle maneuver models to determine a probabilistic match. The predefined maneuvers consisted of a baseline straight and level flight mode, a low G turn mode, a high G turn mode, a climb mode, a climb and turn mode, and an uncontrolled roll mode, which acted as a catch all for irregular vehicle performance. The uncontrolled roll model was the mode used in HITL to detect leader failures.



These graphs depict the Kalman filters outputs for a sample flight path which consisted of a straight and level flight for 15 seconds, followed by a 20 second climb and turn maneuver and ended with a vision sensor fault at 35 seconds. The multi-model probability graphs on top display the raw probability outputs of the filter. These raw probabilities are then processed to determine what vehicle maneuver the lead vehicle is performing. The purpose of this was to remove false model matches derived from the raw data alone. An example of this is seen in the Sensor Fault model shown in the top right graph. The raw data depicts a sensor fault at 15 seconds incorrectly. This false alarm was caused by two factors. The first was the mode switching from straight and level flight to climb and turn, the second was that the vision sensor data was updated at a faster rate than the communication data. The predictive states assume a continued straight and level flight and since the sensor data shows the climb and turn before the communication data it is assumed, for an instance, that the sensors have failed. Once the communication data was updated the sensor fault model probability drops back to near 0 until the actual fault occurred at 35 seconds. The filter also estimates wind based on a comparison of vehicle positions, attitudes, and velocities.





## Kalman Filter HITL Application

- **Filter Code Unable To Run Real-Time At 50 Hz**
  - Designed Flexible Time Steps And Packet Sizes
  - Not Optimized For Speed
- **Modified To Run Parallel To Simulation**
  - Used For Modal Analysis Only
    - Monitor Uncontrolled Roll
  - Estimations Not Used
- **Filter Not Specifically Tuned To The Particular Vehicle**
  - Limits Capabilities
- **Simple Logic Applied For Dynamic Leader Switching**
  - Maintain Pack Position
  - Cycle's Thru Pack Vehicle ID's
  - Filter Delay And Restart To Avoid False Triggering

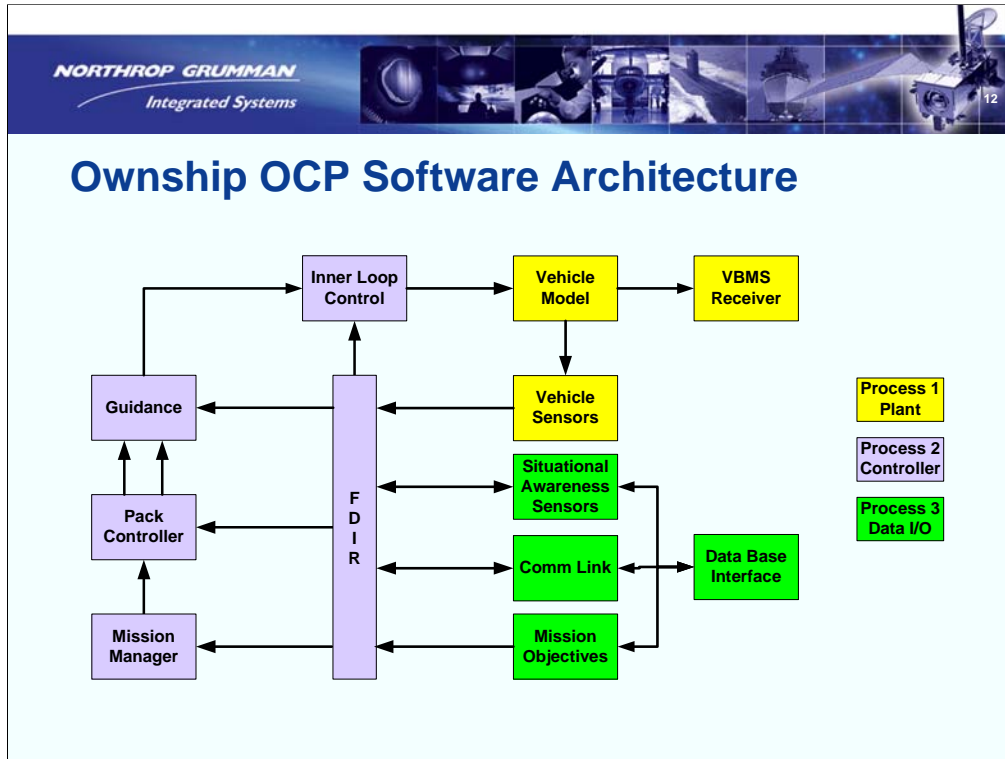
The Kalman filter was developed for robustness not speed under the SEC program and therefore was unable to run at the 50 Hz needed to support the HITL simulation. To compensate for this the filter was slowed to run on a separate 2Hz thread in OCP. While this degraded the usefulness of the filter's estimation data it did provide an opportunity to display the multi rate functionality of OCP. The reduction in rate made the state estimation data from the filter useless for the purpose of driving the station keeping algorithms. Therefore the filter was modified to run parallel to the simulation loop and act only as a fault monitor. Without the use of the Kalman filters state estimation, the communication loss fault scenario became an impossibility. The raw vision sensor data alone could not be used to drive the station keeping algorithm which relies on body rates and velocities which were to be provided by the Kalman filter.

The simulation architecture was designed so that the filter was only run in follower vehicles during pack scenarios and only monitored the lead vehicle. Simple mode transition logic was implemented to use the uncontrolled roll filter model output to reconfigure the vehicles outer loop controller. A fault in the lead vehicle would trigger a reclassification of the lead vehicle to an intruder and the appointment of new leader. The pack reposition themselves around the new leader. The changing of lead vehicles requires the filter to be reset and a delay to be implemented in order to avoid false fault triggering during the transition.

		
Fault Scenarios		
Fault	Test Conditions	Mitigation
<b>Hard Over Surface</b>	<ul style="list-style-type: none"> <li>•Before Intruder Avoidance Maneuver</li> <li>•During <math>t &lt; T_{CPA}</math></li> <li>•At <math>t = T_{CPA}</math></li> <li>•Immediately After Initiation</li> </ul>	<ul style="list-style-type: none"> <li>•SEC Cooperative FDI in Wingman Detects Anomalous Ownship Behavior</li> <li>•Intruder Utilizes Tcas for Avoidance.</li> <li>•Wingman Utilizes Instinct Maneuver Blended with Tcas.</li> </ul>
Ownship-Wingman <b>Comm Loss</b>	Same	<ul style="list-style-type: none"> <li>•Vision Sensor Replaces Communication Data.</li> <li>•Increased Separation</li> <li>•Wingman Utilizes Own Tcas</li> </ul>
<b>Ownship Tcas Loss</b>	Same	<ul style="list-style-type: none"> <li>•Ownship &amp; Wingman Change Roles.</li> </ul>
<b>Rogue Ownship</b> (Ownship Continually Maneuvers to Hit Intruder Or Wingman)	Same	<ul style="list-style-type: none"> <li>•SEC Cooperative FDI in Wingman Detects Anomalous Ownship Behavior</li> <li>•Intruder Utilizes Tcas for Avoidance.</li> <li>•Wingman Utilizes Instinct Maneuver Blended with Tcas.</li> </ul>
<div> <p>Everybody has Travel Alert/ Collision Avoidance System &amp; Talks to Everybody.</p> <p>Wingman Ignores (Tcas) Until Ownship Failure</p> </div>		

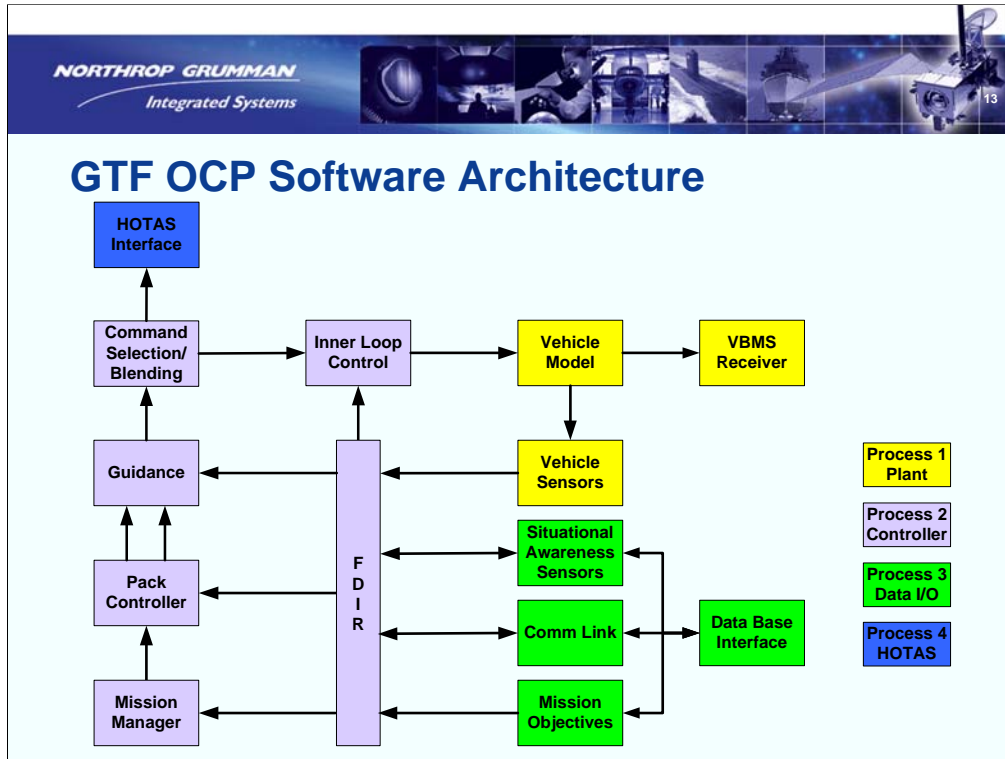
A fault matrix was developed around four separate faults (hard over control surface failure, communication loss between pack, TCAS failure in leader, and a rogue ship). These faults were designed to test the fault detection as well as the reconfigurability and robustness of the control algorithms. Each fault was to be inserted at different times throughout the scenario with respect to a collision engagement. Unfortunately time and cost constraints limited the fault testing to non-engagement scenarios removing the insertion time distinction as well as the Ownship TCAS Loss failure. Also, as mentioned before, the Comm loss scenario was removed due to the inability of the Kalman filter to run at a rate fast enough to provide estimated data to drive the station keeping algorithm.

The remaining two fault scenarios, rogue Ownship and hard over surface, turned out to be very similar. In each case, the lead vehicle was determined to have incurred a failure and the follower vehicles in the pack remove the leader from the pack filter which results in the leader becoming an intruder. The collision avoidance algorithm takes over and maneuvers away from any conflicts.



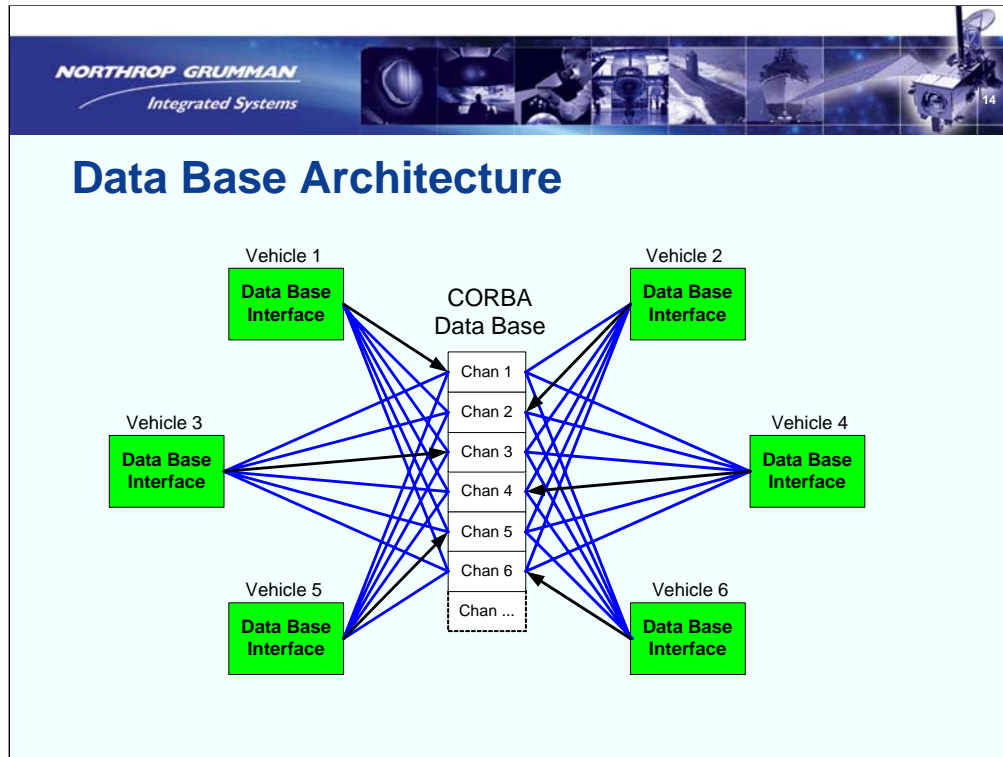
Throughout the program the above architecture was used as a baseline for the Ownship/UAV model. Each box represents an OCP component and are separated into three OCP processes allowing for a distributed simulation to be run on multiple platforms. The first process consisted of the vehicle model as well as the interface to the visualization tool VBMS. The second process contained all of the controller algorithms which were hosted on a PowerPC board. The final process contained the components that represented the external data interfaces. This included the situational awareness sensors data, the communication link data and the mission objectives data. As the program progressed, components were gradually populated in a sequence which supported the three Demonstration/Reviews. The one modification made mid program to this architecture was the addition of a second signal between the Pack Controller and the Guidance components. This second signal was necessary to accommodate the two controller modes (station keeping and waypoint following) of the final demonstration. This modification was facilitated through the OCP Controls API tools without major implication to the simulation.






The Generic Tactical Fighter (GTF) models architecture was based on the same components as the UAV with the addition of a 4<sup>th</sup> process which inputs pilot stick, throttle, and rudder commands. The Command Selection/Blending component was also added to the controller process. This component smoothed the transition between autopilot and pilot commands. The vehicle model and inner loop controller were replaced with a Generic Tactical Fighter model based on NASA Technical Paper 1538. The inner loop controller developed for this paper was based on a piloted vehicle and so the inner loop controller was driven by pilot commanded rudder, and lateral and longitudinal stick forces. Unfortunately the autopilots guidance algorithm for the UAV commanded alpha, beta, and roll rate. In order to create a common interface to the inner loop controller, the pilot's commands were converted to Nz, roll, and yaw commands inside the HOTAS interface component before being sent to the selection/blend component. The original guidance routine was also modified to coincide with the NZ, roll, and yaw command interface. The interchangeability of the pilot and autopilot commands enabled the blending of the two sources which allowed for smoothing of the transition between the two modes. All other components remained exactly the same between the two models.





The CORBA database was based on a publish/subscribe system. Each vehicle would publish its communicated data to a specific section of memory and then subscribes to all other vehicle's memory allocations to access their data. This allowed the creation and destruction of vehicles throughout the simulation as long as the parent vehicle is running. The parent vehicle creates the database and sets the number of vehicles needed to start the simulation and for simplicity purpose was always specified as the Ownship. It creates the database and waits for all other vehicles to register before it sets the run flag which started all the vehicles in the simulation. This operation provided a method to synchronize the individual vehicles. This functionality was later replaced by the implementation of a master OCP trigger which controlled all vehicle's components timing developed by Boeing. The need for a more precise synchronization of the vehicles was found to be necessary for station keeping in tight formation and particularly for use in the Kalman filter where data time stamps were needed. If the time stamps from each vehicle were not synchronized the filter was more likely to fail due to futuristic data created by vehicles with time stamps later than that of the internal vehicle's clock. The end product was a dynamic database external to all simulated vehicles used to pass information between the vehicles.





## Data Base Communication

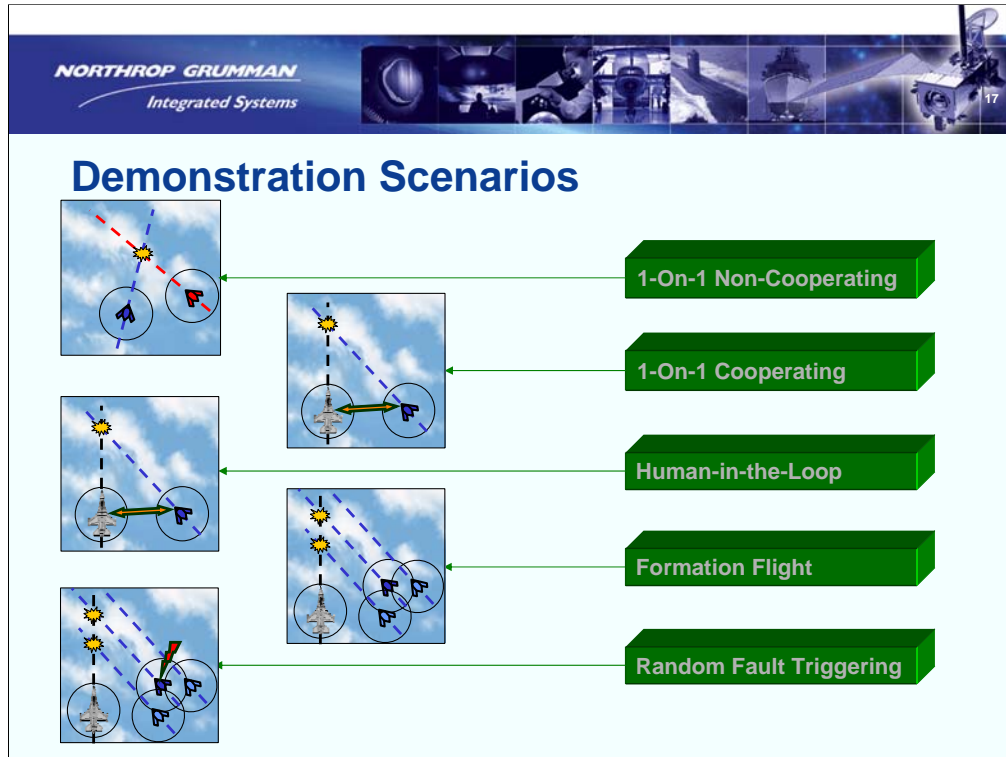
- **Each Vehicle Publishes And Subscribes To A Specific CORBA Data Base Developed By Boeing**
  - Allows Dynamic Creation And Destruction Of Vehicles
  - Allows Synchronization For Repeatability
  - Allows Multiple Instance Of The Same Model To Be Used
    - Ease Creation Of Multiple Vehicle Scenarios
    - Increases Flexibility Of The Scenarios
  - More Representative Of Real World Application

There were many reasons for transitioning to a data base versus the method used in the 1 on 1 cooperating demonstration. In the 1 on 1 cooperative demonstration two vehicles were simulated and communicated via OCP signals. The simulation's architecture had one OCP instance which included both vehicles. Under this system there were a total of seven processes and over 20 components. This turned out to be above the limit which OCP could handle and required a modification of OCP by Boeing to accommodate. Although this enhanced the capability of OCP it was also seen to be a bit unrealistic. The OCP middleware would not be used to communicate between vehicles in real world applications. In addition to this, plans for pack formation flight with an intruder would increase the size of the OCP instance to the point of being unmanageable. Another drawback to this method of communication was the need for a different architecture set up for all different scenarios. There was no flexibility in the number of vehicles of the scenarios. The addition of the data base allowed for multiple instances of a single model to be used to create multi vehicle simulations. Only one architecture was needed and the number of vehicle combination scenarios became limitless. A scenario could consist of as many vehicle as desired. The ability to create and destroy vehicles throughout the simulation was also a benefit which became an apparent during the second demonstration while flying with a pilot in the loop. Whenever the pilot made a mistake and crashed the GTF, the entire simulation died causing a complete restart of all vehicles in the simulation. Under the new architecture, if the GTF crashed, another instance could be started up while the simulation continued without interruption.




## Presentation Outline

- Program Summary
- **Program Results/Lessons Learned**
  - Scenario 1 - 1-On-1 Non-Cooperating
  - Scenario 2 - 1-On-1 Cooperating
  - Scenario 3 - Human-in-the-Loop
- Final Simulation Demonstration
  - Scenario 4 - Formation Flight (Pack Formation)
  - Scenario 5 - Formation Flight (S-Curve)
  - Scenario 6 - Formation Flight With Collision Avoidance
  - Scenario 7 - Formation Flight Random Fault Triggering
- Path Forward



Five demonstration scenarios were chosen to test the HITL simulation. These scenarios were sequenced in a way which built upon each other to aid in the development of the simulation. Each scenario demonstrated added functionality and/or new hardware to the simulation. The first scenario was designed to integrate a vehicle into OCP and run as an all software Windows based demonstration. The second scenario integrated the GTF model into OCP along with the addition of communication capabilities between the vehicles allowing cooperative conflict resolutions. The third scenario added a pilot interface to the GTF model enabling the incorporation of the Infinity Cube simulator in the VACD laboratory. For both the second and third scenarios the Ownship's control algorithms were hosted on the PowerPC hardware. The fourth scenario incorporated formation flight capabilities while the fifth scenario added fault insertion to the formation flight. The end result was a simulation architecture which could host any of the scenarios without changing any code. All specifications needed to move between scenarios were made in a initialization file.

**NORTHROP GRUMMAN**  
*Integrated Systems*

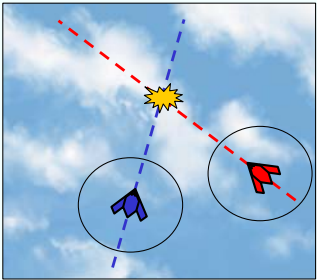


## Lessons Learned

■ **Scenario 1**


- Integrated Low Fidelity UCAV Like Model and Controllers For Use in Subsequent Scenarios.
- Modified TCAS For Autonomous Use.
- Validated TCAS During Non Communicating Engagements.
- Identified TCAS Model Deficiencies.
- Familiarized With OCP Environment

**1-On-1 Non-Cooperating**



The first demonstration scenario provided a foundation for all simulations that followed. The overall vehicle architecture and signals specifications were defined and integrated into the OCP 2.0 environment. A UAV model was integrated into the architecture along with a inner loop controller and a simple 4-D waypoint guidance routine. This combination was set as the baseline Ownship vehicle for all future demonstrations. To achieve the collision avoidance aspect of the scenario an algorithm based on the Traffic Collision Avoidance System was developed. This TCAS algorithm, which normally provides warnings and suggested climb rates to pilots, was modified to directly modify the altitude command being sent to the guidance routine resulting in a successfully performed collision avoidance maneuver. The transition from a passive warning system with a pilot in the loop to an active autonomous controller required some modifications to the TCAS command logic. A time latch was added to the initiation of a TCAS alert as well as a time delay to the disengaging of the TCAS alert. The addition of the initiation latch solved the chatter issue observed in some specific scenarios where the TCAS command would flash on and off for a short period of time while the intruder vehicle was on at the edge of the alert thresholds. The need for a disengage time delay was observed during scenarios where the disengaging of TCAS caused the Ownship to return to its original flight path before a large enough separation was achieved with the intruding vehicle. The delay allowed enough time to achieve sufficient separation between the vehicles before disengaging. With these modifications multiple 1 on 1 scenarios were tested, validating the autonomous collision avoidance maneuver as well as the vehicle models OCP implementation.

**NORTHROP GRUMMAN**  
*Integrated Systems*



## Lessons Learned

- **Scenario 2 & 3**
  - Integrated Generic Tactical Fighter (GTF) and Simple NASA Inner Loop Controller based on Technical Paper 1538.
  - Pilot Interface Added
  - Developed Blending Algorithm to Combine AutoPilot and Pilot's Commands
  - Removed Asymmetry in Data Provided By NASA
  - Developed Autonomous Cooperative TCAS

1-On-1 Cooperating

Human-in-the-Loop

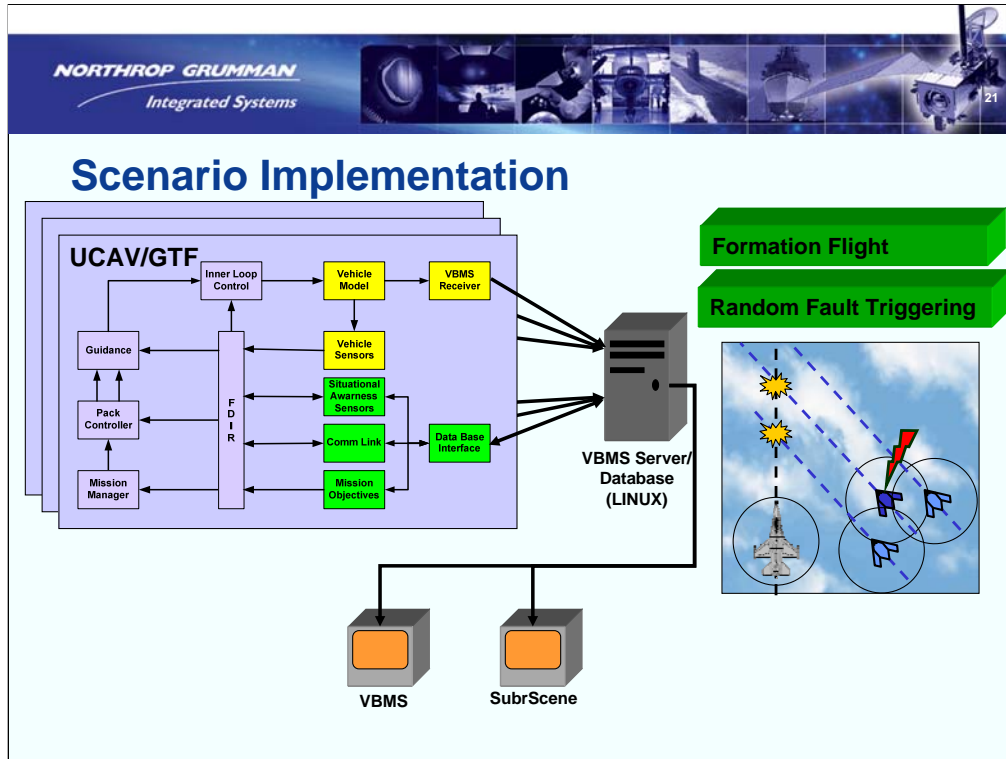


The second demonstration consisted of scenarios 2 and 3. In these scenarios the GTF model was introduced to the OCP environment along with a pilot in the loop capability. The TCAS algorithm matured to include cooperative conflict resolutions and hardware was introduced to the simulation. With the addition of a pilot interface, a component was added to the outer loop to fuse the pilot's commands with the autopilot's. A pilot could fly continuously around the other vehicle causing various TCAS based maneuvering to avoid any midair collisions. A TCAS alert was added to the pilot interface to alert the pilot that a maneuver was necessary to deconflict a possible collision when the autopilot and the autonomous collision avoidance was disengaged. Although the pilot was alerted of a necessary maneuver, he was not required to obey it thus creating a need for the other vehicle to recognize and adjust its own maneuver to accommodate. This functionality along with cooperative conflict resolution was added to the TCAS algorithm for the demonstration of these two scenarios.



## Presentation Outline


- **Program Summary**
- **Program Results/Lessons Learned**
  - Scenario 1 - 1-On-1 Non-Cooperating
  - Scenario 2 - 1-On-1 Cooperating
  - Scenario 3 - Human-in-the-Loop
- **Final Simulation Demonstration**
  - Scenario 4 - Formation Flight (Pack Formation)
  - Scenario 5 - Formation Flight (S-Curve)
  - Scenario 6 - Formation Flight With Collision Avoidance
  - Scenario 7 - Formation Flight Random Fault Triggering
- **Transition**



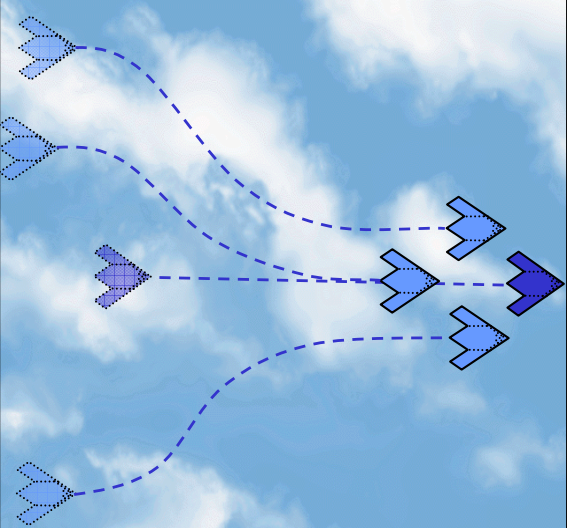
The final demonstration was performed on Linux machines only with the exception of the PowerPC hardware running VxWorks. Only two separate OCP models, one UAV and one GTF, were necessary for all scenarios. Multiple instances of the two models could be created and initialized at start up allowing for an unlimited number of combinations of vehicles for the last two scenarios. Due to limited processors it was determined that a maximum of four vehicles could be run safely (without the worry of frame overruns). Therefore all pack only scenarios consisted of four vehicles flying in a diamond formation and all scenarios mixing a pack with one intruder consisted of three vehicles in a V formation and one intruder GTF vehicle. For the fault scenarios, the ability to set a particular failure to occur at a particular time was provided by the initialization file. Any of the control surfaces (rudder, aileron or elevator) could be set to have a hard over failure at any time. For use in the demonstration only the aileron hard over was implemented.



**NORTHROP GRUMMAN**  
*Integrated Systems*



## Demonstration Scenario #4



**Formation Flight**

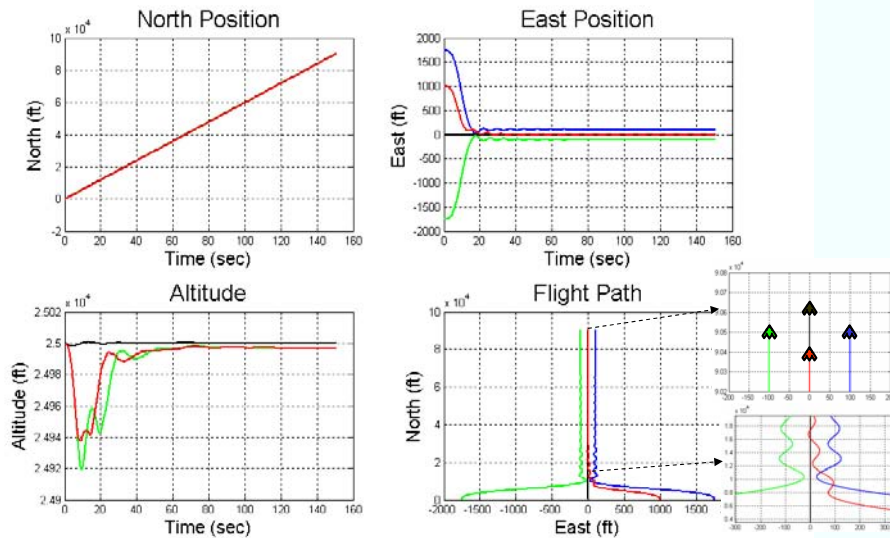
**Vehicles Start Spread Out Across Approximately 1 Mile**

- Station Keeping Algorithm Brings Vehicles Into Diamond Formation
- Formation Determined By Initialization File

Demonstration scenario 4 is the first demonstration of formation flight. The scenario includes four vehicles initially spread out over approximately 1 mile. As the scenario begins, the lead vehicle begins to track its predetermined waypoints while the three following vehicles use their station keeping controller to converge on their position in a tight co-altitude diamond formation. The position in the pack as well as the separation distances are set in an initialization file that is read in at the beginning of the simulation. All positions in the pack are in reference to the lead vehicle.




## Demonstration Scenario #4

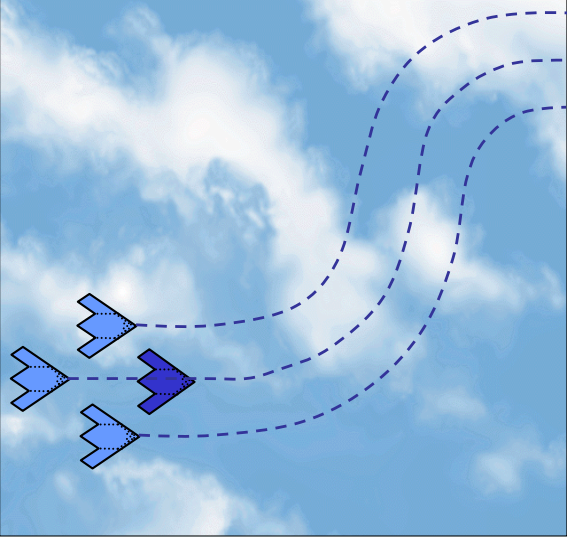


The plots of demonstration scenario 4 show that the vehicles successfully tracked their position in the diamond formation. One issue noted was that the station keeping algorithm gains were loosely tuned in order to allow for the vehicles to converge on their positions when initialized to have huge offset errors. This is apparent in the horizontal tracking oscillations observed in the lower zoomed in display of the Flight Path graph. These oscillations dampened out over a 30 second time period which resulted in a tight diamond formation shown in the final position of the vehicles in the upper zoomed in Flight Path graph. The altitude loss depicted in the first 30 seconds by the follower vehicles are a result of the sharp turning needed to acquire their pack position. Once they are in the vicinity of their correct position their altitude climbs back to the altitude of the leader.

**NORTHROP GRUMMAN**  
*Integrated Systems*



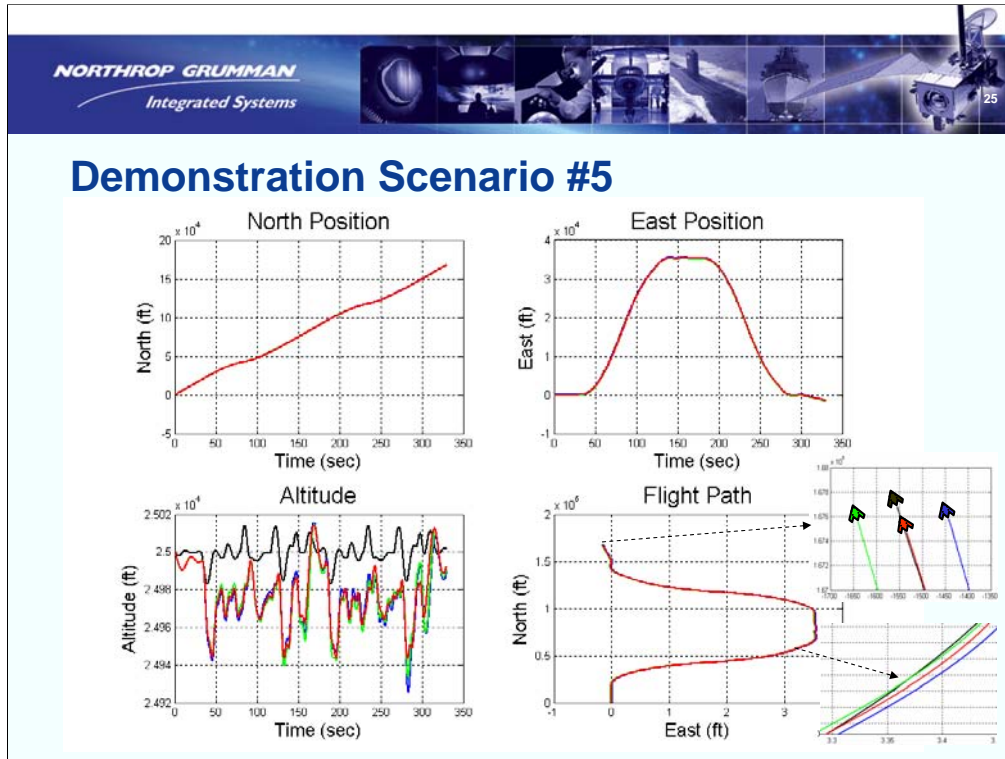
## Demonstration Scenario #5




**Formation Flight**


- Continuation of Scenario 4
- S-Turn Formation Flight
- Vehicles Maintain ~50 ft Separation During Turns
  - Formation Determined By Initialization File
- Followers React to Leader
  - Lag Seen In Turns

Demonstration scenario 5 is a continuation of scenario 4. The vehicles which merged to form the diamond pack formation in scenario 4 continue on to perform formation s-turns. The entire time the vehicles maintain ~50 ft separation from there nearest neighbor. All maneuvering is triggered by the lead vehicle's waypoint file. The pack vehicles have no prior knowledge of the flight plan and simply react to the lead vehicles.

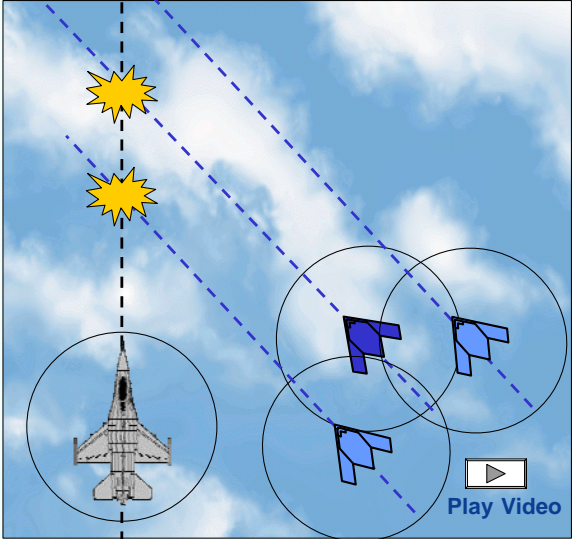


The plots of demonstration scenario 5 show that the pack successfully performs multiple turns while staying in formation. There is apparent lag between the lead vehicle and the reaction of the following vehicles. The lower zoomed in view gives an example of this lag as the lead vehicle actually gets on the left hand side of his left wingman before the wingman reacts and correctly repositions itself. The end result, as in scenario 4, is a close nit diamond formation. Altitude once again bounced around as turns were being performed. The amplitude of this altitude jitter is greater in the follower vehicles due to the fact that they are reacting to the lead vehicles altitude jitter. Also it is noted that during the turns the inside vehicle needs to slow down and the outside vehicle needs to speed up in order to maintain their relative position in the pack. Between scenarios 4 and 5, it is shown that the latitude, longitude, altitude, and speed controllers of the station keeping algorithm are performing fairly well and are robust enough to handle multiple scenarios.





## Demonstration Scenario #6



1-On-1 Cooperating

Human-in-the-Loop

Formation Flight

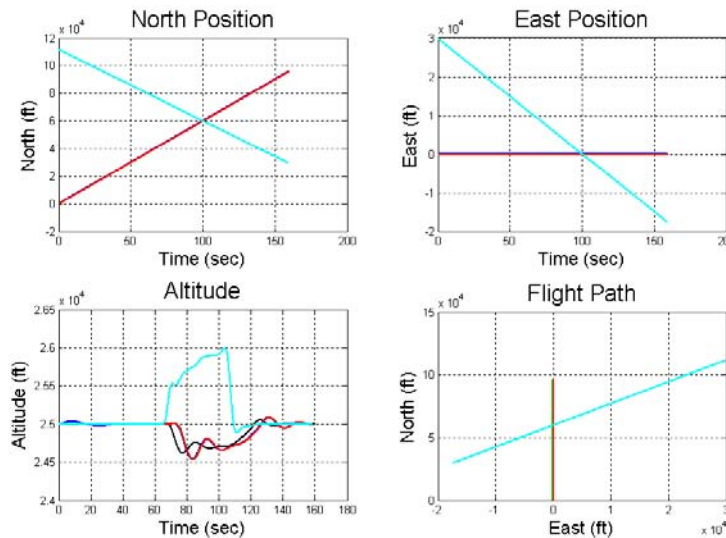
- Leader and GTF Perform Cooperative Collision Avoidance
- Followers React Only To Leader
  - Ignore Own TCAS Alerts
  - Lag Seen In Avoidance Maneuver

Demonstration scenario 6 adds an intruder vehicle to the path of a pack. The pack used in this scenario consists of only three vehicles in a V formation along with an intruder GTF. The reason for this was simple processing power. There is no limitation to the number of vehicles that the final architecture can support. The only limitation is processor time. With more than five vehicles running frame overruns became more frequent and problems arose. This scenario added the 1 on 1 cooperating collision avoidance as well as the human in the loop feature with the formation flight scenario. The GTF receives position data from all of the pack vehicles and vice-a-versa. Only the lead vehicle and the GTF respond to their TCAS command while the followers simply respond to lead vehicle's. The follower vehicles' TCAS continue to track the GTF but ignores the command.






## Demonstration Scenario #6

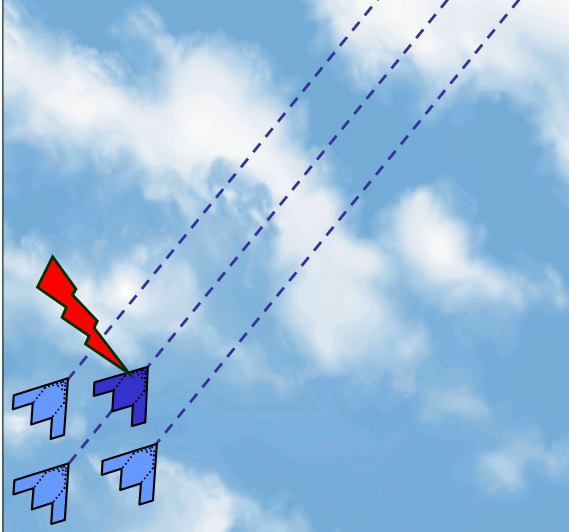


The plots of demonstration scenario 6 show the obtuse engagement scenario. The pack and GTF begin the scenario at the same altitude causing an estimated collision at 100 seconds into the scenario. The GTF's TCAS commands a climb while the UAV leader commands a dive. The GTF's maneuverability allows it to climb faster than the UAV can dive which the TCAS algorithm takes advantage of. This results in the GTF providing about  $\frac{3}{4}$  of the separation needed. The lag seen in scenario 5's turns becomes apparent again in the dive command. The follower vehicles react slowly to the lead vehicles dive. The idea to allow all vehicles in the pack to command their own collision maneuver was a possible solution to this problem but actually created a greater problem of the possibility of conflicting climb/dive commands amongst the pack members. With this in mind the lag response was deemed to be the acceptable choice.

**NORTHROP GRUMMAN**  
Integrated Systems



## Demonstration Scenario #7



Formation Flight

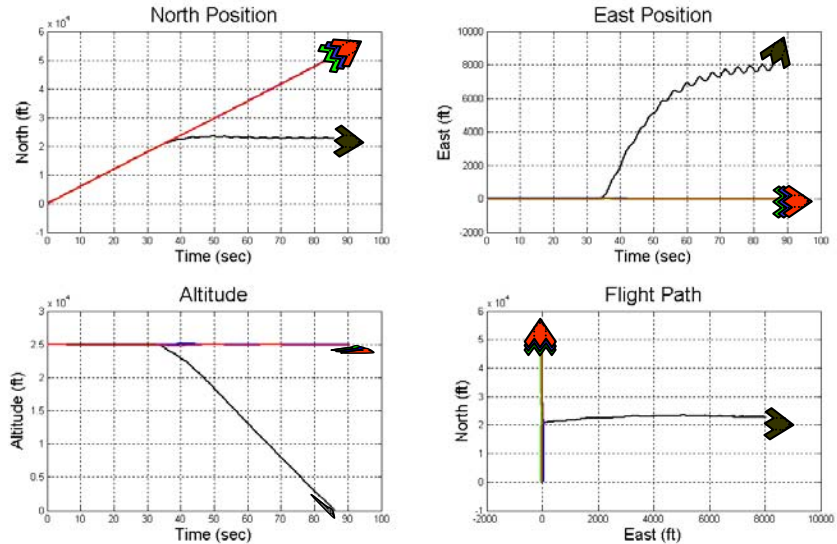
Random Fault Triggering

- **Aileron Hard Over Occurs in Leader at 30 Seconds**
  - Set From Initialization File
- **All Followers Using Kalman Filter to Monitor Leader**
  - Erratic Leader Behavior Triggers Fault Detection
  - Fault Engages TCAS
  - Vehicles Climb To Avoid Leader
  - New Leader Appointed
  - Readjust Formation

Demonstration scenario 7 is a continuation of the pack scenarios 4 and 5 with the addition of a hard over control surface failure. At 30 seconds into the simulation an aileron hard over occurs in the lead vehicle causing it to cut in front of the rest of the pack. Each pack member's Kalman filter independently detects the erratic behavior of the lead vehicle and reconfigure their TCAS filter to include the leader as a intruding vehicle which needs to be avoided. At the same time a new leader is appointed and the new leader uses its TCAS to maneuver the pack out of the way. The pack then reorganizes itself around the new leader and continues on its original waypoint plan. When the new leader is appointed, each follower's Kalman filter is reset to track the new leader. If the new leader incurs a fault, the same process occurs and a new lead vehicle is set. All faults are set by a token in the initialization file which specifies both the type and time.

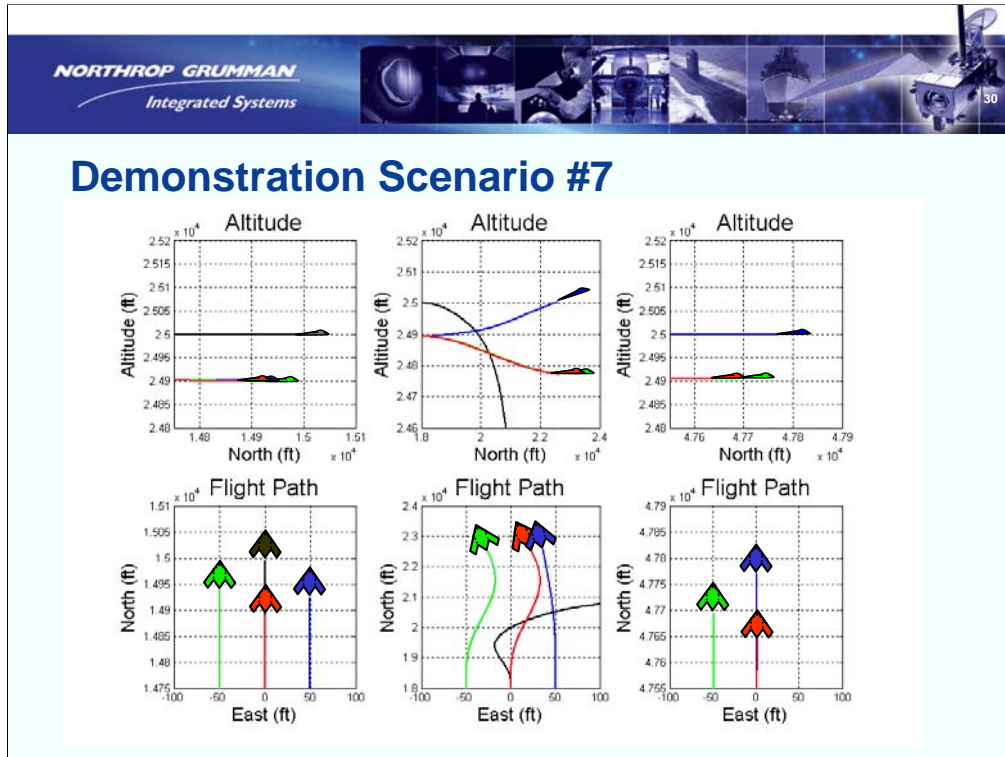


## Demonstration Scenario #7



The plots of demonstration scenario 7 show the pack flying due north. The lead vehicle is at an altitude of 25000 ft while the other vehicles follow at 100 ft below. This altitude separation was set to allow for the lead vehicle to cut in front of the pack causing a situation that the other vehicles must react. If all vehicles were at a co-altitude when the fault occurred the lead vehicle would descend out of the restricted safety area of the new leader before any maneuver could be implemented. By shifting the lead vehicle 100 ft up, the new leader appointed after the fault quickly determines a climb is necessary. The followers then respond to this climb and reposition themselves around the new leader.







A more detailed look at three instances scenario (before, during and after the failure) gives some insight into what occurred in a very short amount of time. The graphs on the left depict a time slice before the failure has occurred. All the vehicles are in a tight diamond formation with the lead vehicle 100 ft above the rest of the pack. In the middle graphs, the failure has just occurred and the leader has cut in front of the right hand wingman. The altitude graph shows the newly appointed blue leader began to climb almost immediately after the lead vehicle began to act erroneous. It is also seen on this graph the the other two vehicles began to dive. This is not the cause of the collision avoidance algorithm but instead the transition from one leader to another. The two remaining followers are attempting to obtain the 100 ft altitude separation and since they were co-altitude when the new leader was appointed they immediately dive. In the flight path graph depicting just after failure it can also be seen that the followers are attempting to horizontally track their new leader. Finally, in the graphs on the right hand side, the end result is depicted. The new leader has positioned itself using the preset waypoints and the followers have repositioned themselves around the new leader.



## Presentation Outline

- **Program Summary**
- **Program Results/Lessons Learned**
  - Scenario 1 - 1-On-1 Non-Cooperating
  - Scenario 2 - 1-On-1 Cooperating
  - Scenario 3 - Human-in-the-Loop
- **Final Simulation Demonstration**
  - Scenario 4 - Formation Flight (Pack Formation)
  - Scenario 5 - Formation Flight (S-Curve)
  - Scenario 6 - Formation Flight With Collision Avoidance
  - Scenario 7 - Formation Flight Random Fault Triggering
- **Path Forward**

## OCP User Development Process

- **Need A Well Defined Static Interface Control Document**
  - Modifications During Development Create Problems
    - GUI BackEnd API Tool Can Introduce Error
      - User Preserve Region Loss
      - Signal Order Switch
      - Created Corrupted Files When Modifying Sequence Signals
      - Renaming Of Some Components Are Not Updated
      - Unable To Remove Triggers Once Built
    - Allows Proper Initialization of Signals
- **Set Up Simulation Outside of OCP**
  - Difficult To Debug Inside OCP
    - Allows Batch Testing At High Speeds
  - Confirm If Code Is Efficient (Capable Of Running Real-Time)
    - OCP Overhead Can Slow Code



Throughout the program, many lessons were learned about the OCP development process. The first and most important is to have a well defined interface control document which remains as static as possible throughout the development. The extra time spent refining the interface before beginning the actual development can save a lot of pain in the future. Although OCP provides an API tool to simplify modifications, there were problems observed after a change was made. Loss of user supplied code which was inserted in an area specified to be a protected preserve region was observed when modifications were made to the interface. The most dangerous problem observed by changing the interface during development was the rearranging of signal member order. This problem created an instance which took many days to diagnose. The problem was that the command sent from the inner loop controller to the vehicle model consisted of four double members in the order elevator, aileron, rudder, and throttle. When a modification was made to a separate signal, the API tool regenerated the code and switched the send order to aileron, elevator, rudder, and throttle. Although the members order was switched there was no warning and no errors showed up during compiling. To diagnose what went wrong, the entire model had to be rebuilt one component at a time to see where the problem appeared. This problem occurred a second time and was caught a little sooner but still took some time track the error. Another aid to diagnose these problems is to first set up the simulation outside of OCP. This allows for high speed batch testing while providing a control to compare OCP results to for verification purposes. It also is much easier to debug code being developed outside of OCP.



## OCP User Development Process Cont.

- **Input One Component At A Time**
  - Allows Individual Testing Of Components
  - Isolates Problems That May Arise During OCP Integration
  - Difficult Due To BackEnd Tool Issues
- **Add Text File Outputs For Each Component**
  - Eases Problem Tracking
  - Ability To Compare Results To Simulation Outside Of OCP
- **Create Wrapper Which Allows Static OCP Code Which Interfaces With User Code**
  - Allow Use of Libraries
  - Implemented In NGC SEC Experiments
  - Ease Switching Algorithms

The next suggested OCP development practice is to build the simulation in steps. Inputting one component at a time allows for individual validation that the code is working properly. Also it was found that external text files outputs recording data throughout the simulation allowed quicker tracking of problems since debugging in OCP is so difficult. The final development process which was discovered during this program but not implemented was the use of a wrapper interface to create a static interface that remained unchanged inside the OCP code while linking in a library that could be compiled separately. This not only allows the protection of source code by use of a library but removes the need to recompile the OCP code whenever changes are made. Instead you would compile the new library and then re-link the OCP process. This technique was used in the development of SEC OCP code used in the flight demonstration.

## Matlab AutoCode Generation Process

- **Embedded Real Time Workshop**
  - Much Easier To Read and Interface
  - Use Interface Definition To Outline Simulink Subsystem
  - Use of Signal Specification Blocks Allows Array Naming in AutoCode
    - External Inputs and Outputs Defined As Structure
    - Default Structure Will Consist of Every Variable
  - Use Tunable Parameters To Ease Adjustments Of Control Gains
  - Matlab 6.5 Auto Code Limitations
    - S-functions Not Supported In Matlab 6.5
    - Multiple Simulink Blocks Not Supported in Matlab 6.5
  - New Matlab 7 Supports Both S-functions, Embedded Matlab and More of the Simulink Blocks
    - Embedded Matlab Greatly Increases Simulink Model Capabilities
    - Define Your Own Simulink Blocks With Matlab Code

This program was one of the first at NGC to use Matlab auto code produced by the Embedded Real Time Workshop toolbox. The code generated was much easier to read and interface to than the Generic Real Time Workshop code. The auto code creates one external input and one external output structure used to interface the code. The interface definition document can be used to outline the Simulink subsystems and with the use of signal specification blocks the auto code interface can be modified to emulate the naming convention. For code which contains parameters that may need to be changed, the tunable parameter function provides easy access to hand modify the values. This becomes useful for cases such as controller gains which may be modified for different vehicles. Instead of needing to auto code two versions of the same controller a tunable parameter can be used to switch between two models different gains.

Matlab 7.0 greatly increased the functionality of the Simulink auto coder. The new version allowed auto coding s-functions as well as embedded Matlab functions which allowed the user to define their own Simulink blocks with Matlab functions.



## Value Added

- **TCAS**
  - Algorithm Used In AFRL/DARPA AFCST Program
    - Follow-on Program SeFAR
  - Basis For Upcoming ACCESS 5 NASA Program
- **OCP Foundation For SEC**
  - Integration Knowledge
    - MATLAB AutoCode Experience
    - Implement AutoCode Generate To OCP Without Modifying Internal OCP Code
- **OCP Maturation**
  - Put OCP Through The Rigors Of Real World Applications
    - Brought To Light Application Issues
    - Boeing Solved Problems And Incorporated In OCP Updates

The HITL program benefited all parties involved. The TCAS algorithm developed under the program was used in the Autonomous Flight Control Sensing Technology (AFCST) AFRL/DARPA program and will continue to be used as a surrogate TCAS in the follow on program Sensing for UAV's Awareness (SeFAR) to be replaced by a commercially available TCAS. The knowledge gained from the development of the TCAS model was also leveraged for the ACCESS 5 program sponsored by NASA/FAA/DoD with a goal of obtaining FAA approval for UAV vehicles to fly in national airspace. The OCP experience and knowledge gained on the HITL program was heavily used in the integration of SEC flight code including MATLAB Simulink auto code. This was especially the case since the NGC technology software for SEC was completely developed in the Simulink environment.

The OCP middleware was put through the rigors of Real World applications. The program began with the use of OCP B2.0 and finished with OCP B2.4+. The plus represents that updates to version 2.4 were present in the final demonstration. The HITL program feels they were partly responsible for distinguishing problems along with suggesting useful features which Boeing was able to incorporate into newer versions of OCP.



## Value Added Cont.

- **Demonstrates Benefits of Middleware**
  - Communication Across Multiple Platforms
  - Portability
  - Ease of Development
    - Debugging In Linux Is Painful (Average Controls Engineer)
- **Expose Non-Software Personnel To Software Requirements And Software Personnel To Real World Applications**
  - Invaluable Knowledge Growth

The HITL program also demonstrated the benefits of a middleware. The OCP platform was used to develop algorithms on a Windows desktop machine. This eased the development process by enabling user friendly debugging provided by the Microsoft Visual Studio environment. The simulation was then easily ported onto Linux boxes and even VxWorks on a PowerPC board all without modification of the source code. Throughout the program, multi platform simulations were demonstrated including all combinations of Windows, Linux, and VxWorks. The program also exposed the non-software savvy controls engineer to software requirements and implementations while also allowing software personnel to get feedback from those who work with real world applications. The program also left in place at the AFRL/VACD laboratory two simulation models with replaceable OCP components to be used for future integration of control algorithms. Also developed was the ability to display real-time simulations in OCP with the VBMS and Subscene visualization tools. Overall the knowledge growth of all parties involved created by this environment was immeasurable.



**NORTHROP GRUMMAN**  
*Integrated Systems*



## Remaining OCP Issues

- **Real-Time Capabilities**
  - Currently Unable To Produce Constant Frame Rate
  - Separate OCP Instances Run at Different Rates
    - Cause Problems In Algorithms
    - Communicated Data Unpredictable
      - Futuristic Time Stamps
      - Multiple Repeated Data
  - Boeing Developed Last Minute Patch
    - Universal Frame Controller
    - Work Around Not Solution
- **OCP Overhead**
  - Code Runs Considerably Slower Inside OCP
- **Environment Differences**
  - Simulation Performance Varies

At the end of the program there were still some issues remaining with the OCP middleware. The real-time capabilities on the Linux platform was observed to be limited. There was an inability to produce constant frame rates. On Windows or Linux operating systems the frame rates have been observed to vary by +/- 1 millisecond on a 50 Hz frame. With the use of separate OCP instances used for multiple vehicles this problem was exposed due to an exaggerated time drift as the simulation ran. The drift became apparent in the internal simulation times, which are calculated by the use of a fixed time step. Time dependent algorithms began to act erroneously. After closer examination of the data it was discovered that each instance's clock was slowly drifting, which was remedied by the implementation of a master trigger for all processes. Though this work around remedied the problem for the demonstration it did not resolve the issue of needing a guaranteed constant frame rate for real time applications. It was also observed that the processing overhead created by OCP limited the capabilities of the demonstration. Code which was tested outside of OCP ran much faster than when implemented inside of an OCP process. To some degree this is expected but it is believed that the current state is not optimal.

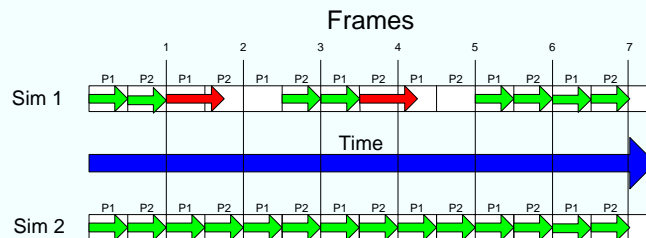




## Remaining OCP Issues Cont.

### Frame Overrun Handling

- Causes Component Skipping
- Component Behavior Triggered By Inputs
  - Overrun Components Don't Send Signal
  - Dependent Components Not Run That Frame
- Need Multiple Behaviors
  - One For Input Signal Storage
  - One For Running Code (Separate Trigger)
  - Constant Frame Rate Issue



When a frame overrun occurs, the next component, which is waiting for an input signal, does not run. In effect, the behavior skips a frame to catch up instead of proceeding with the most current data. Problems arise when there are components of the simulation not controlled by the same frame controller. The example shown contains two OCP simulation instances running independently. The two processes represent the Plant Model (P1) and the Controller (P2). In the second frame of sim 1 the Plant Model has a frame overrun. This causes the controller not to run during this frame. The simulation then picks back up at frame 3 for P2. During this time the other sim is able to meet each frame and therefore by the end of this 7 frame example it is 2 frames behind causing the plant model to be 2 time steps behind. Any communication between the two vehicles is affected by time drifts and repeated data as the simulation progresses.

In Linux, if Process1 crashes or is closed first, a complete reboot of the machine is required. In Windows this only hangs up your machine for a while.

Debugging problems in this environment is extremely difficult due to the lack of repeatability. Each run provides a slightly different frame rate.